OPIC
OFFICE DE LA PROPRIÉTÉ
INTELLECTUELLE DU CANADA

CIPO
CANADIAN INTELLECTUAL
PROPERTY OFFICE

(12) (19) (CA) **Demande-Application**

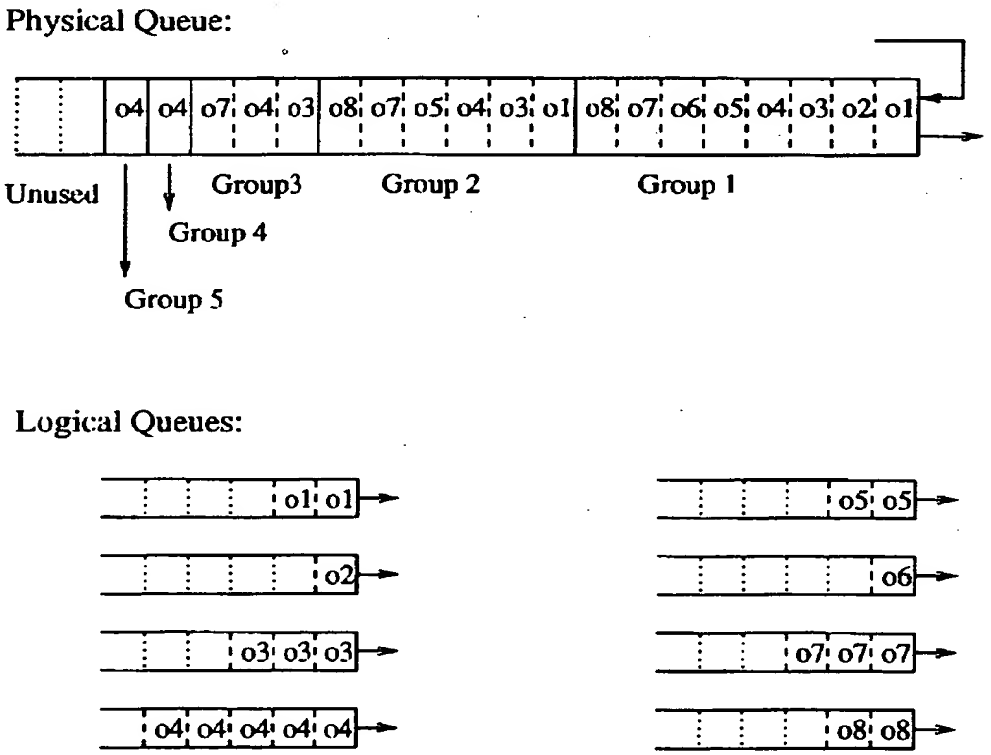
(21) (A1) **2,227,655**
(22) 1998/04/03
(43) 1999/10/03

(72) LEON-GARCIA, Alberto, CA
(72) HASHEMI, Masoud Reza, CA
(71) LEON-GARCIA, Alberto, CA
(71) HASHEMI, Masoud Reza, CA

(51) Int.Cl.^{6} H04L 12/56

(54) **COMMUTATEUR A UNE SEULE FILE D'ATTENTE**

(54) **THE SINGLE-QUEUE SWITCH**



(57) A new switching mechanism for fixed-length data packets such as ATM cells is disclosed that uses a novel buffering element to form a single physical queue in which all the data packets or cells are organized into logical queues destined for different output ports. The buffering element is a generalized sequencer circuit which allows flexible ordering of the cells in each logical queue to achieve any appropriate scheduling algorithm. This single-queue switch implements output buffered queueing and multicasting and achieves full buffer sharing. The number of input and output ports can vary independently without affecting the switch core. The size of the buffering space can be increased simply by cascading the buffering elements. This switch can be used as queue controller in RAM-based switches in which data packets are stored in a RAM memory and a fixed-length data unit represents the packet in the queue controller.

The Single-Queue Switch

Abstract

A new switching mechanism for fixed-length data packets such as ATM cells is disclosed that uses a novel buffering element to form a single physical queue in which all the data packets or cells are organized into logical queues destined for different output ports. The buffering element is a generalized sequencer circuit which allows flexible ordering of the cells in each logical queue to achieve any appropriate scheduling algorithm. This single-queue switch implements output buffered queueing and multicasting and achieves full buffer sharing. The number of input and output ports can vary independently without affecting the switch core. The size of the buffering space can be increased simply by cascading the buffering elements. This switch can be used as queue controller in RAM-based switches in which data packets are stored in a RAM memory and a fixed-length data unit represents the packet in the queue controller.

The Single-Queue Switch

Abstract

~~A new switching mechanism for fixed-length data packets such as ATM cells is disclosed that uses a novel buffering element to form a single physical queue in which all the data packets or cells are organized into logical queues destined for different output ports. The buffering element is a generalized sequencer circuit which allows flexible ordering of the cells in each logical queue to achieve any appropriate scheduling algorithm. This single-queue switch implements output buffered queueing and multicasting and achieves full buffer sharing. The number of input and output ports can vary independently without affecting the switch core. The size of the buffering space can be increased simply by cascading the buffering elements. This switch can be used as queue controller in RAM-based switches in which data packets are stored in a RAM memory and a fixed-length data unit represents the packet in the queue controller.~~

Background of the invention

Field of Invention

The present invention in general relates to the high-speed packet switches used in communication and computer networks to switch data packets arriving in a plurality of input lines to a plurality of output lines. The present invention can be used in switches in broadband communication networks in general and broadband integrated services network switches such as in ATM (Asynchronous Transfer Mode) or IP (Internet protocol).

In the said switches data packets or cells arriving from input ports are stored, queued, scheduled, and switched to output ports. In output-buffered switches arriving cells are switched and put in separate output buffers according to their destinations. Data packets are queued and scheduled in the said output buffers and are sent to the output ports accordingly. The circuit of the current invention can replace the bank of output buffers in the said switches.

In RAM-based switches, arriving cells are stored in a RAM memory, and a queue controller maintains the output queues. The circuit of this invention can be used as queue controller in the said RAM-based switches, such as RAM-based ATM switches.

Discussion of Previous Art

The design of an ATM switch architecture with low hardware complexity and high performance in throughput, buffer usage, and quality of service such as delay and cell loss, has been a challenge for the past few years. Output queueing, complete buffer sharing, and scheduling capabilities for a guaranteed quality of service, are considered as major characteristics to be achieved by a successful ATM switch.

Output buffering guarantees maximum throughput and buffer sharing reduces the amount of required buffering space. While many of switches use output buffering, not all of them achieve a high degree of buffer sharing. Full buffer sharing can be achieved in RAM-based shared buffer switches, but it comes with a large overhead, in terms of hardware complexity to control the buffering mechanism and to keep track of the cells and the queues as well as the free spaces in the buffer.

In terms of queue scheduling, most switches are inflexible in terms of the scheduling algorithms they can implement. The scheduling mechanism has to fit within the constraints of the switch architecture, and changes in the scheduling scheme require extra hardware, or even a revision in the overall hardware design. For RAM-based switch architectures, the output queues were originally FIFO (First-In-First-Out). To provide priority-based queueing, a group of FIFOs can be used for each output line. Still, only a limited number of priority levels are possible per output line. US patents 5,274,642, 5,406,556, and 5,440,553 entitled "Output-buffered Packet Switch with a Flexible Buffer Management Scheme," show how an output buffered packet switch with priority can be built by storing packets in a shared RAM memory and by using a queue controller to control the reading of packets to the output lines.

Scalability is another major issue. It is not always possible to increase the number of input and output lines easily without reconstructing the whole switch. This is very clear in Banyan-based switches. In RAM-based switches the buffering section is independent of the number of I/O (input/output) lines but still increasing the number of lines requires that extra queues be implemented in the control section of the switch.

Increasing the buffering space also is not an easy task in many switches. In RAM-based switches the buffering space can be increased by using larger RAMs, but in most of the other switches additional changes in the hardware are required.

Full buffer sharing in the control memory is very important in RAM-based switch architecture. Typically the queue controller consists of separate FIFO queues per output line. The FIFO queue(s) for each line contain minicells that represent cells that have been stored in the main RAM memory. When a minicell reaches the head-of-the-line and has highest priority, the corresponding cell is read out from the RAM to the corresponding output line. For large number of input lines and priority levels, the buffering space used to store the mini-cells is very high and can even be comparable to the space used to store the cells themselves. This is because this FIFO space is not shared among the queues, unlike the main RAM which is shared among the output queues. The buffer requirement for the controller part can be reduced with a factor of 1/NP, in which N is the number of output lines and P is the number of priority levels for each output line, if the buffer is shared by the output queues [H. Kondoh et al, "A Scalable Nonblocking Shared Multibuffer ATM Switch with a New Concept of Searchable Queue," Proceedings of International Switching Symposium '95, pp 278-282, 1995].

To achieve this goal Kondoh proposed a switch architecture in which the cells (mini-cells) destined for different output lines all are put in the same physical buffer. The cells are identified by a tag attached to them showing their destination. The resulting architecture cannot operate at high speeds, it has no flexibility in providing scheduling schemes, and the architecture is not scalable. The reason for all of these shortcomings is the following. In this architecture cells are put in the common buffer in the order of their arrival. The buffer is a searchable queue in which a common search circuit and a common data bus is used to find the first matching cell for each output line at each cell time. The length of the queue is limited because of the bus structure and because the search mechanism requires flag passing. The sharing the search circuit among the output lines means that search takes place sequentially. The hardware is directly dependent on the number of output lines (N), and the latency is also related to N regardless of the traffic load. Finally there is no flexibility in scheduling; Only two-priority scheduling is possible with a very inflexible mechanism.

The present invention develops a cell switch by generalizing the cell sequencer introduced in [M.R. Hashemi and A. Leon-Garcia, "Input Buffering and Back-Pressure Mechanism in ATM Switches," Proc. of SPIE Conference, Vol. 2917, pp 398-409, Boston, Nov. 1996] and [M.R. Hashemi and A. Leon-Garcia, "A General-Purpose Cell Sequencer/Scheduler for ATM Switches," Proc. of IEEE Infocom, April 1997]. Prior work on cell sequencers has focussed on the management of single queues.

In the sorting circuit described in the US patent 4,991,134 entitled "Concurrent Sorting Apparatus and Method Using FIFO Stacks," and all similar circuits cited in that patent, information enters the queue from the tail of the queue and has to travel to the head of the queue through sorting steps before being able to leave the queue. Also the sorting is based specifically on the magnitude of data blocks. The sequencer in the present invention sorts the blocks of data based on magnitude or logical comparisons. The logic of comparison is arbitrary and can be implemented in the logic section of each unit. Also, data blocks enter from the head of queue. As a result, the latency between the arrival of a new cell and departure of a cell from the queue is comprised solely of the delay in the first unit. The rest of the units operate concurrently and in the background.

The sorting circuit described in the US patent 5,504,919 entitled "Sorter Structure Based on Shiftable Content Memory," and US patent 5,313,579 entitled "B-ISDN Sequencer Chip Device," use a bus structure to sort the data. All sorting circuits which use a bus structure including the above-mentioned ones, suffer from a high latency in operation at each data arrival because of the time required for propagation of the result of the comparison at each unit to the rest of the queue. A new cell can enter only after the settlement of the previous cell in the queue. Therefore the arrival rate cannot be high. Also the comparison is based on magnitude and no manipulation of data is possible.

The sequencer used in the current invention is not based on the bus structure. The new data can enter immediately after the previous one. The departing cell is determined immediately and the rearrangement of the rest of the queue is done concurrently with new arrivals. Also logical comparison and manipulation of information in the cell tags is possible.

Summary of the Invention

The objective of this invention is to achieve fully shared output queueing in ATM switches and in the queue controller of RAM-based switch architectures. It is also an objective of this invention that cells can be scheduled flexibly in the output queues. This invention is based on a novel buffering device which can be used as the core of a switch or as a queue controller for RAM-based switches. The said device, called single-queue switch, queues the cells or minicells of output ports in logically separate output queues while the said logical output queues all are physically in the same buffer.

The single-queue switch is output buffered and offers full buffer sharing. The number of input and output lines can vary without affecting the switch core. The buffering space can be increased by simply cascading the buffering elements. The scheduling mechanism implemented in the switch is software programmable and is flexible enough to realize a wide range of scheduling algorithms.

The architecture of the switch is based on the cell sequencer architecture introduced in [M.R. Hashemi and A. Leon-Garcia, "Input Buffering and Back-Pressure Mechanism in ATM Switches," Proc. of SPIE Conference, Vol. 2917, pp 398-409, Nov. 1996] and [M.R. Hashemi and A. Leon-Garcia, "A General-Purpose Cell Sequencer/Scheduler for ATM Switches," Proc. of IEEE Infocom, April

1997]. In the sequencer, cells are stored and queued in a shift-register-type buffering element. The structure of the buffer allows the cells to travel in the queue in pipelined fashion until they find their appropriate place in the queue based on their priority level, age, or any other consideration that can be handled by a flexible comparator element. In the normal mode, the cell at the head of the queue exits the sequencer shortly after the arrival of each new cell.

The present invention will show how the above sequencer can be generalized to distinguish groups of cells with some common characteristics and arrange them in a logical queue. It is then possible to schedule the transmission among these logical queues by appropriately organizing the logical queues in the buffer. The organization of the cells in the logical queues and the logical queues in the buffer is achieved by adding appropriate tags to individual cells before they enter the sequencer. As a result, the scheduling scheme that is implemented can be changed by translating the desired algorithm into an appropriate tagging method.

In particular in this invention, we extend the above sequencer architecture to introduce a new and totally different approach to ATM switching. We introduce an ATM switch architecture which uses a single sequencer-based queueing buffer for all of the output lines. Cells destined for different output lines are organized in different logical queues within the same physical buffer. These logical queues are interleaved into the single queue. Each of the logical queues can have other logical queues inside it to provide flexible scheduling capability.

Cells are inserted into the single queue switch one by one. This is similar to the RAM-based switch where the cells are written into the RAM one by one.

A major characteristic of the single queue switch is its simplicity. The core hardware of the switch is a shift-register-type buffering element. The buffering element has a modular structure and is composed of a collection of an arbitrary number of simple 1-cell-size buffering units which contain a very simple logic part.

The RAM-based switch architecture can be modified by using the single queue switch as its controller/scheduler. Cells are stored in RAM and mini-cells containing a pointer in the tag section are sent to the controller instead. The resultant switch benefits from the advantages of both the single queue architecture and the RAM-based switch architecture while relaxing the queueing overhead and scheduling restrictions of current RAM-based architectures. The hardware of the single queue switch is also minimized in this way by buffering minicells instead of cells.

The single queue switch architecture can be further modified to enable it to provide multicast transmission. The modified single queue switch can then be used as a controller/scheduler in a RAM-based switch architecture.

Detailed Description of the Preferred Embodiment

The single queue switch architecture is based on the sequencer architecture described in [M.R. Hashemi and A. Leon-Garcia, "Input Buffering and Back-Pressure Mechanism in ATM Switches," Proc. of SPIE Conference, Vol. 2917, pp 398-409, Boston, Nov. 1996] and [M.R. Hashemi and A. Leon-Garcia, "A General-Purpose Cell Sequencer/Scheduler for ATM Switches," Proc. of IEEE Infocom, April 1997]. The following is a brief review of the sequencer architecture and its function. The sequencer consists of a chain of buffering units. Each buffering unit can accommodate one cell, which is a fixed-length block of data that includes a tag. Cells can travel from one unit to the adjacent units in forward and backward directions (Figure 1). The sequencer is designed to allow the cell with higher priority to move forward in the queue. At each cell-time t a new cell enters the queue from the head of the queue. The cell is compared to the cell at the head of the queue and the one with higher priority is sent out as the winner. The other cell, the loser, is sent one step back inside the queue where it is compared to the cell in that unit. Again, the winner is sent forward, to occupy the unit at the head of the queue, and the loser is sent backward to the next unit in the queue. This procedure is repeated at every unit spreading into the buffer, like a wave, until it reaches the last unit. In this way, the winner of the unit i is always forwarded to the unit i-1 (the forward path), and the loser to the unit i+1 (the backward path) as is shown in Figure 1.

The next new cell enters the buffer just after the previous one, generating a new wave. The events inside the sequencer are exactly synchronized. Therefore, whenever the new wave arrives at unit i, the outcome of the previous wave is already in the unit and the comparison can start immediately. Each arriving cell moves backward into the queue until it finds its right place in the queue pushing other cells back in the queue.

A blocking capability is implemented in the sequencer, so that the cells can enter the queue even when cell is allowed to exit the sequencer. To realize this capability, in case of blocking the winner in the comparison in each unit remains at the same unit instead of being forwarded to the adjacent unit. The loser is sent backward as before.

The basic feature of the sequencer architecture is the ability to have several logical queues within the same physical queue. For example, each logical queue can represent cells belonging to a given priority class. Additional functions can be implemented within each logical queue such as age control, discarding over-aged cells and so on. In this invention we exploit the fact that it is also possible to build sublogical queues within each logical queue. In particular, we not only put the logical queues destined for the same output port in the same queue, but we also merge the queues of different output ports into a single physical queue.

Scheduling and Programmability

In comparing the cells in each buffering unit within the sequencer, the priority is determined by the tag of each cell. Cells are tagged before entering the sequencer. Tagging can be done in different ways to achieve different purposes. The tag values are obtained from the desired scheduling algorithm which can satisfy the Quality of Service requirements of the connections or flows. Various scheduling algorithms including weighted fair queueing and generalized processor sharing are discussed in the book by Keshav [S. Keshav, "An Engineering Approach to Computer Networking," Addison-Wesley, 1997] and [M.R. Hashemi and A. Leon-Garcia, "Implementatin of Scheduling Schemes using a Sequencer Circuit," Proceedings SPIE, Dallas, November 1997].

The basic mechanism in the sequencer is flexible enough to perform different scheduling schemes by simply changing the tagging algorithm. As an example, weighted fair queueing and priority-based algorithms can be handled by the same hardware. In the first case, the tag reflects the service time of a cell, which is determined upon its arrival, and the cells are served in the order of their service times. In the second case, the tag reflects the priority level of a cell and the cells are served in the order of their priority levels.

The tag can be composed of several distinct fields, each controlling a relative aspect in sequencing the cells. For example, an age field can be used, in conjunction with the priority field, to arrange the cells with the same priority in the order of their ages (Figure 2).

In addition programmable options can be implemented in the hardware and controlled by flag bits in the tag. For example, discarding over-aged cells, aging, and joining the priority and age fields together can be enabled or disabled by using flag bits in the tag field of each cell.

As a further step, regarding the advances in the field of Field Programmable Gate Arrays (FPGA's), programmable logic can be used for the logic section of the units in the sequencer, allowing a wider range of selections for the function provided by this section. A general chip can be manufactured and later be customized for specific switch designs by appropriately programming the logic section. The comparator logic, length of the age and priority fields, position and function of other fields in the tag and other features can be selected in this way.

Single Queue Switch Architecture

Figure 3 shows the overall block diagram of the single queue switch system. At this point we consider a switch with equal number of input and output lines (N), all of them with the same speed. As in any other cell switch, appropriate header translation is required after the cells enter the switch system. In the next stage and in a round robin fashion, all of the input lines are scanned and their cells, if any, are sent to the tagging unit one after the other. Note that if a single cell requires T seconds to arrive in an input line, then the single queue switch must be capable of accepting a new cell every t=T/N seconds. This is similar to the RAM-based switch where the cells are written into the RAM one by one. This could become a bottleneck. In the RAM-based switch the speedup required to overcome this bottleneck is achieved by handling the cells in parallel format. The same approach is used in the single queue switch to write the cells into the tagging unit and the buffer. Therefore, cells are first changed into parallel format before being scanned by this section.

The operations in the tagging unit and in the single queue switch are pipelined. After the tagging unit the cells enter the single queue switch which is a modified sequencer. The latency in the sequencer is very small and equal to the delay in the operation of the first unit only. When a cell enters the sequencer the winner of the first unit leaves the sequencer after a short delay while the loser will experience a longer delay during its journey inside the sequencer. The departing cells then enter the output stage, where operations such as multicasting can be performed. Finally the cells are sent to the output lines in serial format, after the (parallel-to-serial conversion) p/s section.

In the tagging unit, a tag is added to the header of the cell, based on the selected scheduling scheme and also on the information held in a look-up table for the virtual channel or flow to which the cell belongs. A local controller programmed by a processor which is connected through the interface logic, controls the tagging. In this way, software can determine the operation of the controller and the scheduling scheme which it implements. The controller can also receive information from the output stage and the sequencer regarding the last serviced cell and the queue length to implement scheduling schemes such as generalized processor sharing, and self-clocked fair queueing.

Switching Mechanism

As we saw before, the sequencer architecture has the capability of organizing the logical queues within the same physical queue. In the single queue switch architecture we use the same capability to interleave the queues of different output lines into the same physical queue. The cells destined for a given output line or port are said to belong to the same logical queue. The interleaving mechanism is as follows. The sequence of cells is divided into groups. The first group contains the first cell of each logical (output) queue. The second group contains the second cell of each logical (output) queue and so on. Within each group the cells are placed in order of increasing logical output queue number. An example of the interleaved sequence of the cells is shown in Figure 4. If the logical queue of one of the output lines has only k cells, no place is reserved for that output in groups k+1 and up. In this way, in each group only outputs which have cell for that group will occupy a place. This does not mean that new cells cannot be inserted in the groups. For instance, in the above example, if a new cell comes in for the mentioned output, it will be inserted in the appropriate place in the group k+1, pushing the rest of the cells one step back in the queue. As a result, in this switch all of the buffering spaces are shared by all of the input and also output lines. Each unoccupied place in each group can be used by others.

The arrival of a cell to a full buffer can trigger the discarding of a cell. Figure 5 illustrates this situation, assuming that the output is blocked for a duration of t seconds. If the queue has the situation shown in (a), and a new cell for O3 (output number 3) comes in, the result will be as (b), where the last cell of O4 is pushed out because O4 has already occupied all the existing places in the buffer. Normally, at each cell time T each output with a non-empty queue will have a departure. As a result, one group of cells will leave the queue every T seconds. Therefore, in this case the final situation in the above example will be as in (c).

A cell pushout occurs only if the physical queue is already full, the outgoing group is not full and the number of its cells is less than the number of arriving cells at that time-slot, and the newly arriving cells do not include the missing cell in the outgoing group (Figure 6).

In an N by N switch, up to N cells enter the switch at each cell-time. On the output side, one group leaves the switch every cell-time T. The cells are sent to their related output lines using a synchronous round robin scheme. The switch can have up to N departures at each cell-time T. This will happen only when the HoL group has one cell for each output line, otherwise, one or more of the outputs will not be utilized during a time-slot t. The time to transfer the group remains constant. The output will be idle during the turns of the missing cells. Obviously, the queue will not move forward during this time period necessitating the use of the aforementioned blocking mechanism.

Grouping Mechanism

In this section we will explain the grouping mechanism in more detail. For simplicity, at this point we assume that the output queues are simple FIFO queues. In the next section we will extend the mechanism for priority-based queues.

Based on the operation of the sequencer which we explained earlier, each cell entering the sequencer carries a field in its tag which indicates the output line which is its destination. Each cell carries also two flags, each represented by one single bit in a different field of the tag. Let's call these flags "X" and "Z".

The "X" flag of a cell in the queue indicates whether the cell is the last cell of a group or not. Groups are recognized and distinguished from each other by setting the "X" flag of the last cell of each group to "1". The "X" flag of a cell which occupies an empty unit is always set to "1". This in fact starts building the groups at the beginning or adding new groups to the bottom of the queue later. From now on we will refer to the "X" flag of a cell already settled in a unit as "X'" to distinguish it from the "X" flag of a transit cell entering the unit in the backward path. We will use the same notion for other flags as well.

To show that cells can be grouped as we explained before, we start with a simpler method in which the order of the cells within the group is not fixed. We also assume that the output is blocked so the queue does not move forward and therefore no cell leaves the queue.

In a group, a newcomer cell first looks for the cells with the same address tag. In this method, such a cell could be anywhere in the group. Therefore, the cell examines all the cells in the group from the beginning of the group until it reaches a cell with the same address or the end of the group, whichever comes first. The cell always examines the "X'" flag of the cells in the units to recognize the last cell of the group.

If the cell meets a cell with the same address during its journey in the group, this means that it should continue its journey to the next group, to find the bottom of its related logical queue. The cell memorizes this by setting its "Z" flag to "1" (Figure 7 , branch III), and continues its flight to the end of the group which is recognized by examining the "X'" flag of the rest of the cells in the group (Fig. 7, branch II). When this cell meets a cell with "X'" flag set to "1" , it resets its "Z" flag to "0" and goes to the next unit which is the beginning of the next group (Fig. 7, branch I).

If the cell meets a cell with the same address which itself is the last cell of the group, it doesn't need to set its "Z" flag to "1" because it is already at the end of the group. The cell goes to the next unit without any change in the flags (Fig. 7, branch IV).

If the cell meets a cell with a different address which is not the last cell of the group, it goes directly to the next unit to continue its search (Fig. 7, branch V). If the cell does not meet a cell with the same address in the group and reaches the end of the group, it means that the cell must stay at this group as the last cell of its related logical queue. In order to do this, the "X'" flag of the last cell of the queue is reset to "0", and the "X" flag of the new cell is set to "1" instead (Fig. 7, branch VI); Then the cell continues its flight to the next unit and stays at that unit as the last cell of the group, as well as the last cell of its related logical queue. To guarantee this operation, a cell whose "X" flag is "1" always occupies the unit (Fig. 7, branch VII).

In this method, a new cell always enters the unit at the bottom of the first group that does not have a cell for the same destination and the cell in that unit , which is the first cell of the next group, is sent out. This cell, pushed back from the beginning of the group, eventually settles at the bottom of the same group in a similar way. This procedure repeats as a chain phenomenon until the end of the physical queue.

Now we consider the method in which the cells inside the groups are in order, that is in increasing order from 1 to N , based on their destination addresses. We still assume that the queue is blocked. In this method a flying cell not only looks for the cells with the same destination address, but also examines whether their address fields are greater than its own address field or not. Since the addresses are in increasing order, if the cell with "Z" equal to "0" meets a cell with a larger address without meeting a cell with the same address, it means that there is no cell for the same destination of the transit cell in this group. The cell must stay at that unit, as the last cell of its logical queue. The cell occupies the place sending out the cell in that place. This will start a chain phenomenon again which will push all the cells one step (one unit) back. In all other cases the procedure is similar to the previous method. Figure 9 shows the required modification in figure 7, to cover this method.

Finally, we consider the case that the queue is not blocked (the normal case). In this case a new cell may enter the queue at the middle of a group (the Head-of-Line (HoL) group). In order for the algorithm to be consistent, the "Z" flag of the cell entering the queue should be set to "0" or "1" respectively if the cell is entering the queue before or after the time-slot belonging to its related output line. This is done easily because the output stage of the switch operates synchronously and it has an assigned time-slot for each output line during which the cell destined for the output line departs or the queue remains idle (blocked) if there is not such a cell in the group. The "X" flag must always be "0" when a cell enters the queue.

Priority based logical queues:

The logical queues explained so far were simple FIFO queues. An arriving cell in those queues was always be put at the end of its logical queue. In this section we will show that priority-based queueing can also be realized in this architecture.

Despite the fact that all of the logical queues are interleaved in a single physical queue, according to the mechanism explained in the previous section, each logical queue can be viewed as an independent virtual queue built on an independent sequencer. Since the basic structure of the hardware is the same as the sequencer structure shown in Figure 8, we can easily implement the same functions as in the sequencer on top of the interleaving mechanism. Therefore consider a single logical queue with sequencing facilities as shown in Figure 8. To provide the desired sequencing capability, all that is needed is an appropriate tagging algorithm. Obviously, the sequencing mechanism should take effect after a transit cell finds the unit which belongs to its logical queue.

We can summarize the whole procedure after a cell enters a unit, as follows. The address field of the cell is compared to the address field of the cell in the unit. If the two addresses are different the procedure continues as explained before. If the two addresses are the same, the priority fields of the two cells are compared to each other before any other action. If the transit cell is the loser, the procedure continues as before. If the transit cell is winner, it enters the unit and pushes its cell out, at the same time the "Z" flag of the loser (now in the backward path) is set to "1". Also if the loser was the last cell of the group ("X'" was "1") now its "X" is set to "0" and the "X'" of the winner (now in the unit) is set to "1" instead. After this point, the situation is similar to the one in the previous section, so the same procedures are repeated in subsequent units. The whole procedure is shown in figure 9 and summarized in figure 10.

As an optional feature, a third flag, "Y" , can be used to indicate the last cell of a logical queue. This flag can be useful in implementing more complicated queueing strategies. The "Y'" flag is set to "1" if the cell is the last cell of the logical queue. This is the cell with "Z" equal to "0" that meets a cell with higher address or the last cell of the group without meeting any cell with the same address in the group (Fig. 7, branch VI). The "Y'" flag of a cell is reset to "0" whenever a new cell with the same address meets it.

Properties of Single Queue Switch

The single queue switch architecture allows full buffer sharing for the output queues. In this architecture, the buffering spaces are not dedicated or reserved for a specific queue. The buffer is filled from top to bottom. No space is left empty . When a new cell arrives, it is inserted in its right place and the rest of the queue is pushed one step back to use the next empty unit from the buffering space.

Since the logical queues are filled independently, a shorter queue can grow independently of the situation in the other queues, as long as there are empty units in the buffer. If the buffer is full, a shorter queue still grows independently but in this case, other queues which occupy the rest of the physical queue, will lose their cells in the bottom of the queue. So we can conclude that the buffer sharing mechanism will exhibit the push-out property if the buffer is full (Figure 6).

In an N by N switch, to have a throughput sufficient for handling all of the traffic coming from the input lines, the internal speed of the single queue has to be N times the speed of each line. This can be achieved using a faster clock in the buffer. However, if the maximum possible speed in the switch is limited, the required speed up can be achieved by using a parallel path structure. If the internal speed of the buffer is R bits per second and the path width is W, the throughput of the switch will be RW bps. This throughput can correspond to N input lines each with a speed equal to r bps, so that R.W=N.r .

The throughput situation discussed here is similar to the one in the RAM-based shared buffer switch architecture, with a major difference that in RAM-based architecture the internal speed of the switch is limited by the RAM access time while in the single queue switch it is limited by the latency in the comparison and shift operations performed by each unit.

The number of input and output lines in this architecture can vary independently of each other. The core switch architecture is completely independent of the number of input and output lines if the internal speed is higher than the larger of the aggregate input speed and the aggregate output speed.

The mechanism for grouping is totally independent of the number of the input lines; The aggregate flow of incoming cells enter the queue regardless of their origin. Regarding the output lines, the length of the group will change if the number of output lines changes. But, the group size will be automatically adjusted to the number of the output lines. Only the output stage needs to be modified to handle the added lines.

The size of the buffer (maximum queue length) can easily grow by adding the additional buffering units to the switch. There is no impact on the operation of the switch. The speed of operation is completely independent of the buffer size.

Figure 3 shows the overall block diagram of the single queue switch system. The input and output stages are required in all switching systems in various forms and with almost the same complexity. The tagging section is the only major difference in this switch. This section can be merged with the header translation section which usually uses a look-up table, by using a larger memory and extended header attachment. An additional local processor would be required in conjunction with the controlling processor to determine the tags dynamically for the cases that use complicated scheduling algorithms. As usual, the switching system relies on an attached computer, for required communication processes, as a controller. This controller, run by software, can program the local hardware driven processor based on the selected scheduling algorithm.

The core switching section of this switch, the sequencer, is very simple in structure in comparison to the other switches. The structure is modular and is composed of a chain of similar small units. Each unit is made up of a cell storage section and a logic circuit which controls the comparison operation. The logic section is not considered as a major factor, from the VLSI technology point of view. As it is shown in figure 10, the comparison logic can be implemented using two comparators and a combinational circuit with the outputs of the comparators and "Z", "X", and "X'" as its inputs and the direction selector signal and new values of "Z", "Z'", "X", and "X'" as its outputs.

The storage section is a W-bit wide shift register-type memory whose length is L/W in which L is the total cell length including the header and W is the internal data path width. This section has a simple structure because addressing is not required for the buffers. Data bits or words are simply shifted in the buffer which could be implemented by customising the available memory technology. This section, which in fact is the dominant part of the switch, can be minimized, making possible implementation of larger circuits in VLSI, by combining the switch with a RAM memory to store the main body of the cells. We will elaborate on this in the next section.

Application: RAM-Based Switches

The single queue switch architecture can be used as the centralized controller in RAM-based switches. In this case the original cells are saved in the RAM and the mini-cells are sent to the queue controller instead. The mini-cells are composed of the tag of the cell and a pointer which contains the address of the original cell in the RAM (Figure 11).

The size of the buffer in each unit and the width of the data path are reduced to the length of the mini-cells. The advantage of the smaller size of the buffering section of the units is the feasibility of very large number of units in the buffer, since most of the area in each unit is covered by the storage section. Also, using a smaller number of data lines reduces the complexity and also the number of pads required to make the chips cascadable. A major advantage of using the single queue switch as the queue controller in RAM-based switch is the capability to have any number of priority levels and the flexibility to use a wide range of scheduling algorithms while still benefiting from the RAM as the major storage resource in the switch. It is worth noting that in this new architecture illustrated in figure 11, there is no need to have additional hardware to control the virtual queues of chained cells for output lines or their priority levels. Also the buffer-sharing in the single queue switch controller reduces the required buffering space by a large factor as discussed in the introduction.

Multicasting Scheme and Copy Mechanism

In RAM-based switches multicasting is typically provided by including an additional queue for multicast cells. In the multicast single-queue architecture proposed here, an extra logical output queue is implemented in the sequencer to replace the multicasting output buffer of conventional RAM-based switches. The implementation of the new queue is achieved easily by using a dummy output line address for this queue. Since the order of the cells of logical queues in each group is based on their output numbers, the address number 0 is selected for the multicast queue, so the multicast cells are placed at the beginning of each group (Figure 12). Therefore, the real output lines are numbered starting from 1.

In the single-queue switch, at each cell time, the head-of-line group is read out and its cells are sent to their related outputs in a synchronous manner. If the slot of one of the output lines is empty, the queue is not read out (the queue is blocked). In this way synchronization is maintained in the output stage. In the multicasting architecture the scheme is different for the multicast cells. A multicast cell is stored in a register at the output controller of the switch at the beginning of a cell time as shown in Figure 13. A bit-mapped list of the cell's destinations is loaded into another register at the same time as shown in Figure 13.

During the dedicated time slot of each output line, if the output line is among the destinations of the multicast cell, the unicast cell for the output line is compared to the multicast cell in the output controller. If the multicast cell has higher priority, it is copied to the output line. The other cell has to remain in its output queue. This cell is fed back to the queue in the same way that a new cell enters the queue. According to the grouping algorithm, the cell will find its right place in the queue again. If the unicast cell has higher priority, it is sent to the output and a unicast copy of the multicast cell is sent back into the unicast queue instead. The output port number field in the tag of the copied cell is changed but the pointer and the priority indicator fields are not changed. A multicast flag bit in the tag is marked to indicate that the minicell is a copy of a multicast cell as shown in Figure 14.

In this copy mechanism the cell is copied only at the time of departure, and it replaces a departing cell so that it does not require additional buffering space. The output controller consists of a comparator and a selector which determines the direction of the multicast and the unicast cells, based on the result of the comparison.

In the multicasting single-queue switch, since at each time slot t a new cell can enter the queue, the primary backward path cannot be used for the feedback cells, unless the speed of operation in the queue is high enough to accommodate both the new arrivals and the feedback cells. For example, assuming that the sequencer's internal speed is twice as much as its input line's speed, the two entries can be multiplexed in the same path using a simple selector circuit (Figure 15).

This speedup approach can be used to add multicasting capability to a single-queue switch. No changes in the single queue switch design are required. All additional functions are handled by the output controller. The only requirement is an additional flag bit in the minicell to mark copies of multicast cells in unicast queues.

We now consider the use of the destination list and memory management for multicast cells. Upon arrival of a multicast cell in the switch, the list of the destinations is loaded from the look-up table which holds the setup information for the connections. This information is written in the table during the connection or flow setup phase, and is used for translation of connection or flow identifiers and for assigning the physical output line address tag for each cell in the switch fabric. For multicast connections, the table contains a bit-mapped list of the destinations.

At the same time that the cell is stored in the main memory and a minicell is generated for it, the list of destinations is written into a separate memory, using the same pointer that is used to save the cell in the main memory.

The multicast queue has one departure at each cell time. During the multicast queue's time slot, the pointer of the minicell entering the output stage is used to load the bit-mapped list from the control memory to the destination list register associated with the multicast register in the output controller (Figure 13).

During the cell time T, all of the destinations of the multicast cell indicated in the bit-mapped list are served either by sending a copy of the cell to the output port, if it has higher priority, or by sending a copy of the cell to the (unicast) output queue, if it has lower priority. In either case, the minicell is cleared from the output stage at the end of the cell time. If the cell is sent to all of its output lines, its buffer in the memory is freed to be used by new cells. If the minicell is copied back to one or more unicast queues, the cell is still kept in the memory. The multicast register in the output controller has a companion counter register to count the number of times the cell is copied. For each copy of the cell the counter is increased by one. At the end of the cell time, the content of the counter is written in another block of the control memory again using the pointer of the minicell.

Later, at the time that one of these unicast copies arrives in the output stage, it is recognized by its multicast flag bit. The pointer of the minicell is used to load the copy count for this cell in another counter which is associated with the unicast cell register in the output controller (Figure 13). The counter is reduced by one if the cell is sent to output port. If the counter becomes zero, the cell is discarded and the memory is freed. Otherwise the content of the counter is returned back to the control memory.

minicell is copied back to one or more unicast queues, the cell is still kept in the memory. The multicast register in the output controller has a companion counter register to count the number of times the cell is copied. For each copy of the cell the counter is increased by one. At the end of the cell time, the content of the counter is written in another block of the control memory again using the pointer of the minicell.

Later, at the time that one of these unicast copies arrives in the output stage, it is recognized by its multicast flag bit. The pointer of the minicell is used to load the copy count for this cell in another counter which is associated with the unicast cell register in the output controller (Figure 13). The counter is reduced by one if the cell is sent to output port. If the counter becomes zero, the cell is discarded and the memory is freed. Otherwise the content of the counter is returned back to the control memory.

Claims

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A grouping method for representing the contents of N output queues in terms of a single physical queue:
   (a) wherein data blocks in each output queue are arranged according to tag values that indicate relative priority;
   (b) wherein the data blocks in the physical queue are arranged in groups;
   (c) wherein group i contains the ith data block, if any, of each of the output queues;
   (d) wherein data blocks in a group are arranged in order of increasing output queue number;
   (e) wherein the tag for each data block contains a single-bit flag, denoted by X, such that the data block at the end of each group has X=1 and all other data blocks in the said group have X=0;
   (f) wherein the "X" flag of an empty data block is set to "1".

2. The sequencer structure comprising:
   (a) a plurality of series-connected buffering units configured to store a plurality of ordered data blocks;
   (b) wherein the first buffering unit has:
      (i) an output link from which data blocks exit the sequencer;
      (ii) an input link from which data blocks enter the sequencer;
      (iii) wherein input data blocks contain a tag which includes an X flag bit set to "0", a Z flag bit set to "0", a field specifying its output queue number and an optional priority field;

(c) a forward link connecting each buffering unit to the buffering unit immediately before the said unit; a backward link connecting each buffering unit to the buffering unit immediately after the said unit;

(d) wherein each buffering unit comprises:

(i) a buffer that can accommodate one fixed-length block of data that includes a tag;

(ii) a register that holds a new fixed-length block of data;

(iii) comparator logic that accepts as input the tag values of the two fixed-length blocks of data, and using the grouping method of claim 1, determines one of the said blocks to be the winner and the other said block to be the loser, modifies X and Z flag bits in the tags of the data blocks according to Figure 10;

(e) wherein each buffering unit directs the winner data block in the forward link and the loser data block in the backward link if a blocking signal is inactive;

(f) wherein each buffering unit directs the winner data block to its buffer and the loser data block in the backward link if a blocking signal is active.

3. A single queue switch apparatus comprising:

(a) a plurality of input ports and output ports;

(b) an input port controller for transferring input data blocks from the input ports to a tagging unit in synchronous and round robin fashion;

(c) the sequencer structure of claim 2;

(d) a tagging unit which accepts input data blocks, attaches a tag to each data block according to a given scheduling algorithm and according to the grouping method of claim 1, and transfers said data blocks to the sequencer;

(e) an output controller for transferring data blocks from the above said sequencer to the output ports in synchronous round robin fashion;

(f) wherein the output controller generates a blocking signal whenever the group does not have a data block for the current output port in the round robin schedule.

4. A RAM-based cell switch apparatus comprising:

(a) RAM memory for the storage of data blocks;

(b) a queue controller consisting of the sequencer of claim 2;

(c) an input port controller that:

(i) transfers data blocks to RAM memory;

(ii) creates a minicell for each data block, wherein the minicell is comprised of a tag and a pointer to the storage location of the data block;

(iii) wherein said tag includes an X flag bit set to "0", a Z flag bit set to "0", a field specifying its output queue number and an optional priority field;

(iv) wherein said input controller transfers the minicells to a queue controller consisting of the sequencer of claim 2;

(d) an output controller which reads out data blocks from RAM memory to the output ports in synchronous round robin fashion according to the group of minicells produced by the queue controller;

(e) a memory controller which maintains a list of available buffer spaces in the RAM for storing new data blocks.

5. A modified grouping method for representing the contents of one multicast queue and N unicast output queues whereby:

(a) the grouping method of claim 1 is used wherein

(i) output port number 0 is used for the multicast data blocks;

(ii) output ports number 1 to N are used for the unicast data blocks.

6. A multicast output controller apparatus comprising:

(a) a register for the storage of the multicast data block that is transferred from an associated sequencer;

(b) a register for the storage of a bit-mapped list of the multicast data block's destinations;

(c) a first counter register which is initialized to the total number of destinations for a multicast data block and which is decremented each time the data block is transferred to an output port;

(d) a register for the storage of a unicast data block that is transferred from an associated sequencer;

(e) a second counter register which contains the number of remaining transmissions for a unicast data block that was created from a multicast data block;

(f) a comparator and a selector circuit which determine the direction of the multicast and the unicast data blocks, based on the result of a comparison of the priority fields of the associated tag values, wherein

(i) the winning data block is transferred to the output ports;

(ii) the loser data block is returned to the associated sequencer if the data block was unicast;

(iii) the loser data block is copied to the associated sequencer with a modified tag so that the output port number is changed to the unicast port number and a multicast flag bit is set;

(g) control memory for the storage of copy counters and destination lists.

(7) A RAM-based cell switch apparatus comprising:

(a) RAM memory for the storage of data blocks;

(b) a queue controller consisting of the sequencer of claim 2;

(c) an input port controller that:

(i) transfers data blocks to RAM memory;

(ii) creates a minicell for each data block, wherein the minicell is comprised of a tag and a pointer to the storage location of the data block;
(iii) wherein said tag includes an X flag bit set to "0", a Z flag bit set to "0", a field specifying its output queue number and an optional priority field according to the modified grouping method of claim 5;
(iv) wherein said input controller transfers the minicells to a queue controller consisting of the sequencer of claim 2;

(d) the multicast output controller of claim 6 wherein the output minicell of the said controller directs the reading out of data blocks from RAM memory to the output ports in synchronous round robin fashion;
(e) a memory controller which maintains a list of available buffer spaces in the RAM for storing new data blocks.

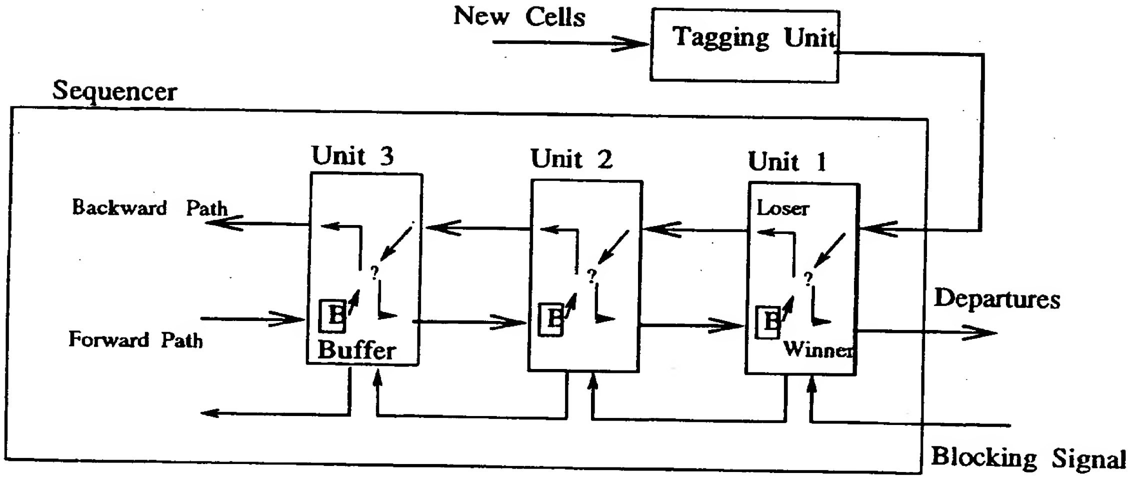


Figure 1: The sequencer architecture

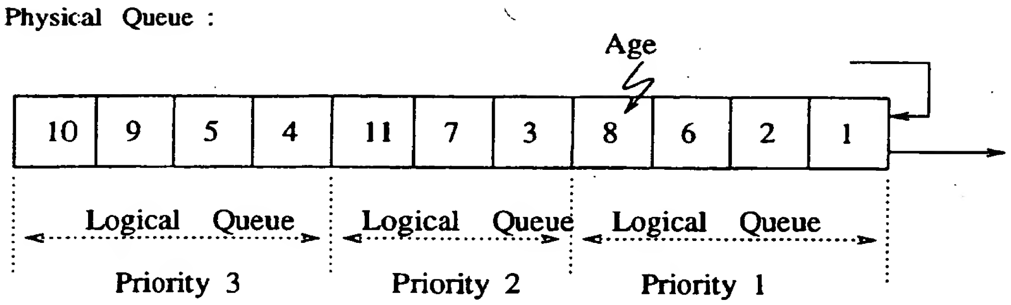


Figure 2: Priority based logical queues

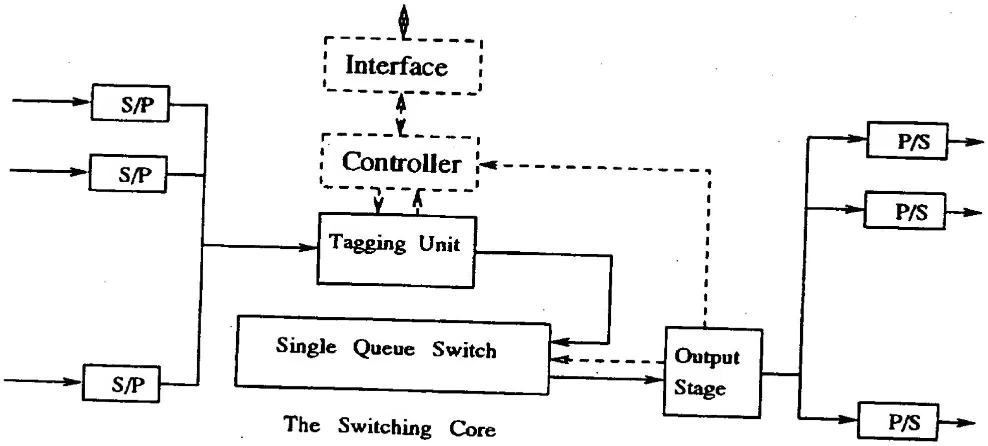


Figure 3: Single Queue Switch System

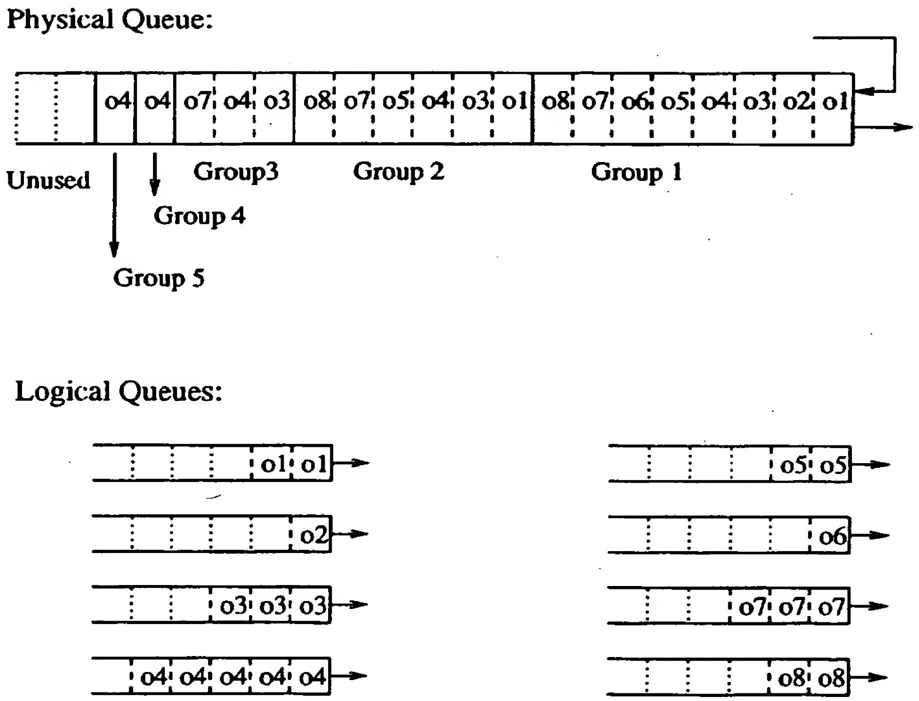


Figure 4: Grouping of cells for $N = 8$

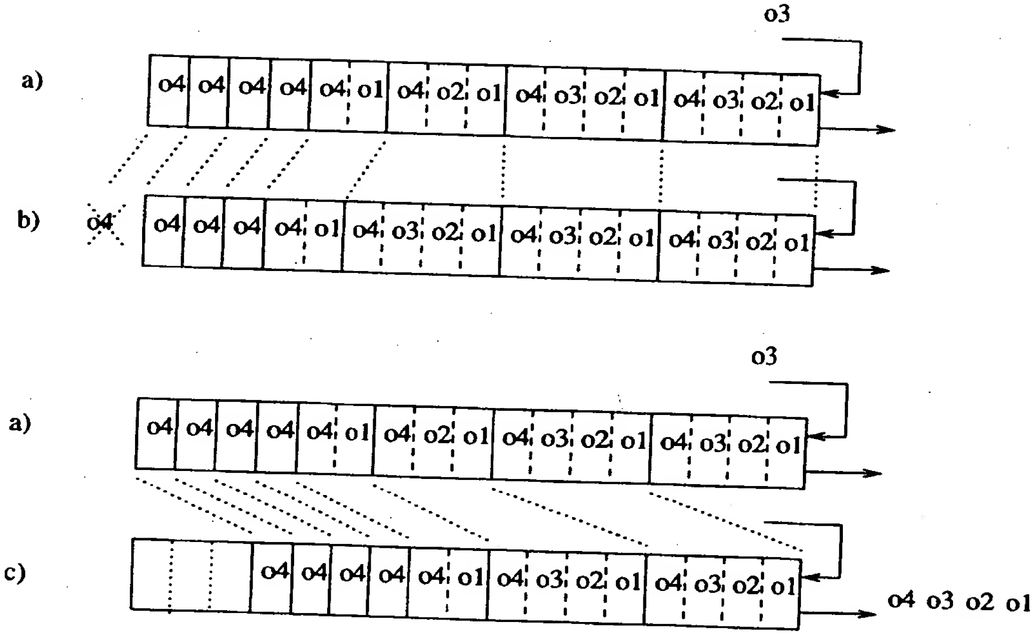

o3
a)
o4 o4 o4 o4 o4 o1 o4 o2 o1 o4 o3 o2 o1 o4 o3 o2 o1
b)
o4
o4 o4 o4 o4 o1 o4 o3 o2 o1 o4 o3 o2 o1 o4 o3 o2 o1
o3
a)
o4 o4 o4 o4 o4 o1 o4 o2 o1 o4 o3 o2 o1 o4 o3 o2 o1
c)
o4 o4 o4 o4 o4 o1 o4 o3 o2 o1 o4 o3 o2 o1
o4 o3 o2 o1

Figure 5: Cell insertion

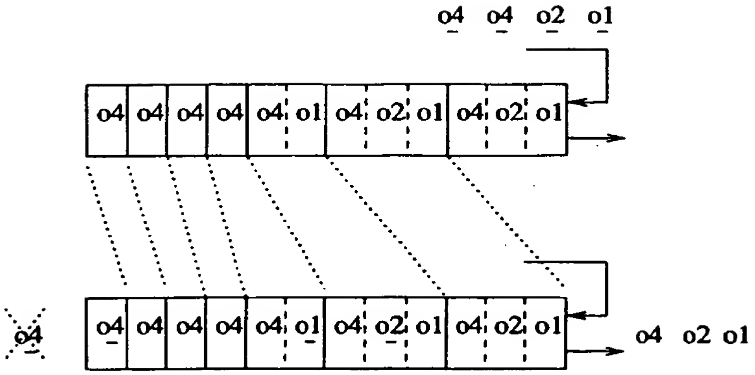

o4 o4 o2 o1
o4 o4 o4 o4 o4 o1 o4 o2 o1 o4 o2 o1
o4
o4 o4 o4 o4 o4 o1 o4 o2 o1 o4 o2 o1
o4 o2 o1

Figure 6: push out

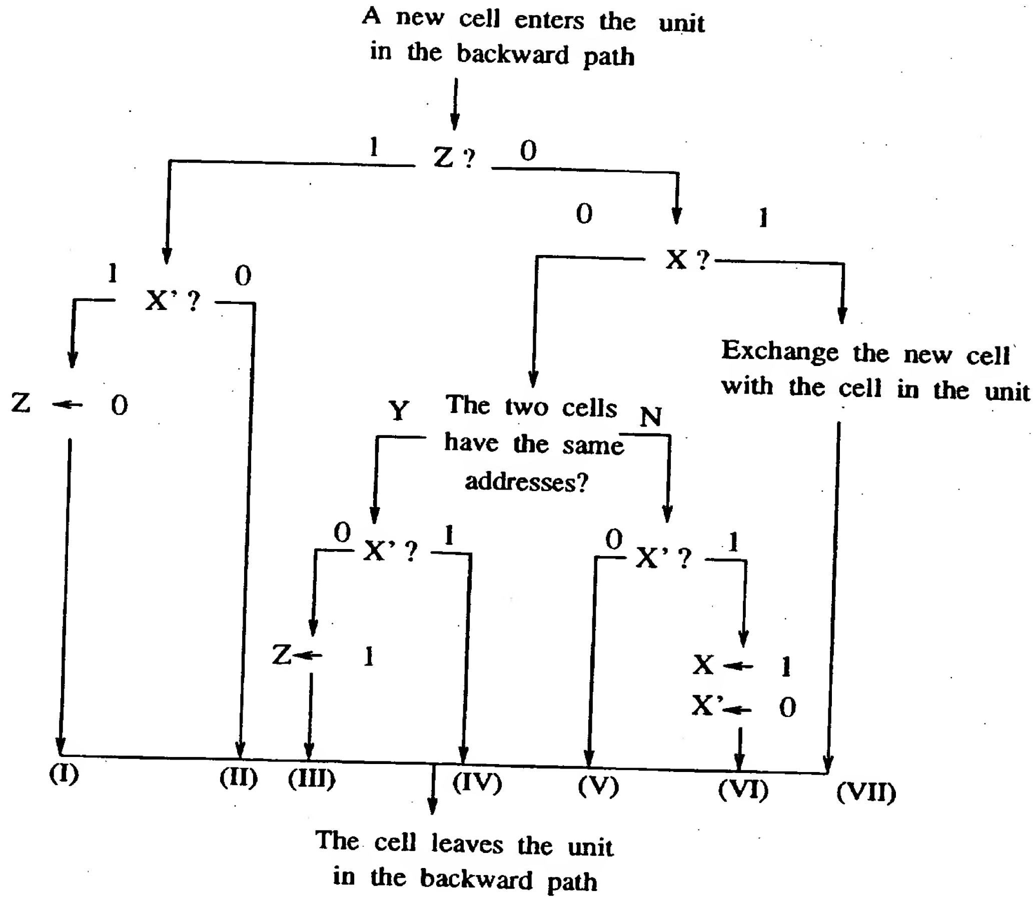

A new cell enters the unit in the backward path
1
Z ?
0
0
X ?
1
1
X' ?
0
Z ← 0
Exchange the new cell with the cell in the unit
Y
The two cells have the same addresses?
N
0
X' ?
1
0
X' ?
1
Z← 1
X ← 1
X'← 0
(I)
(II)
(III)
(IV)
(V)
(VI)
(VII)
The cell leaves the unit in the backward path

Figure 7: The comparison logic in the simplest case. The items with (') refer to the cell which is in the unit, the other items refer to the cell which enters the unit.

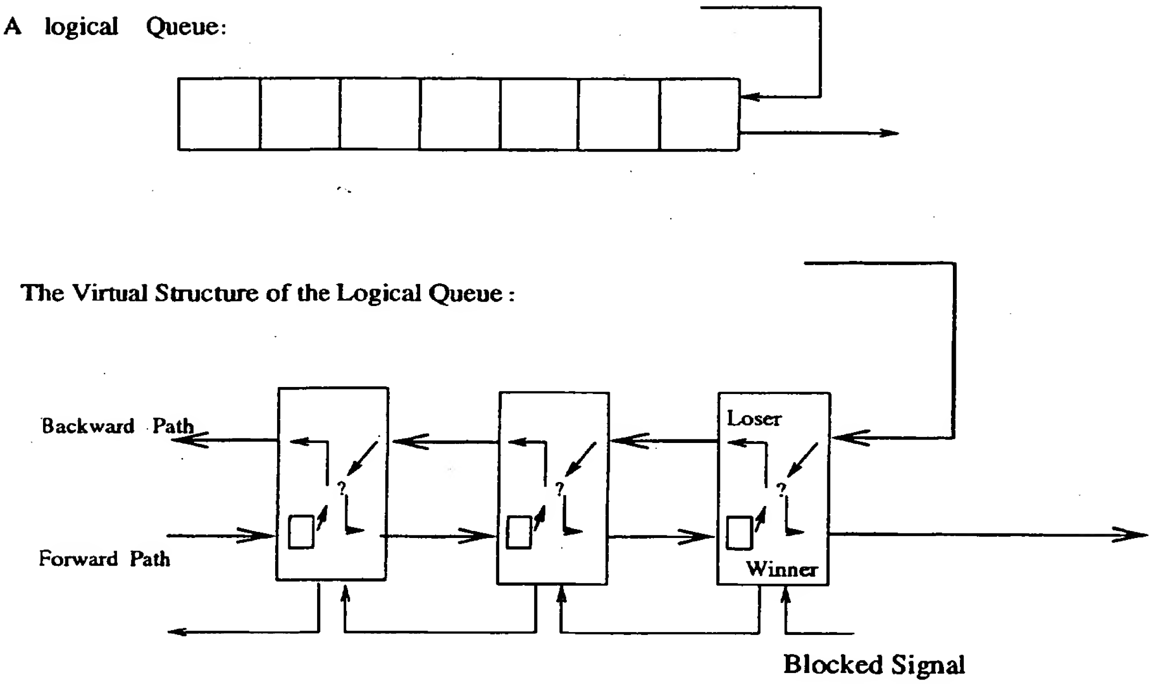
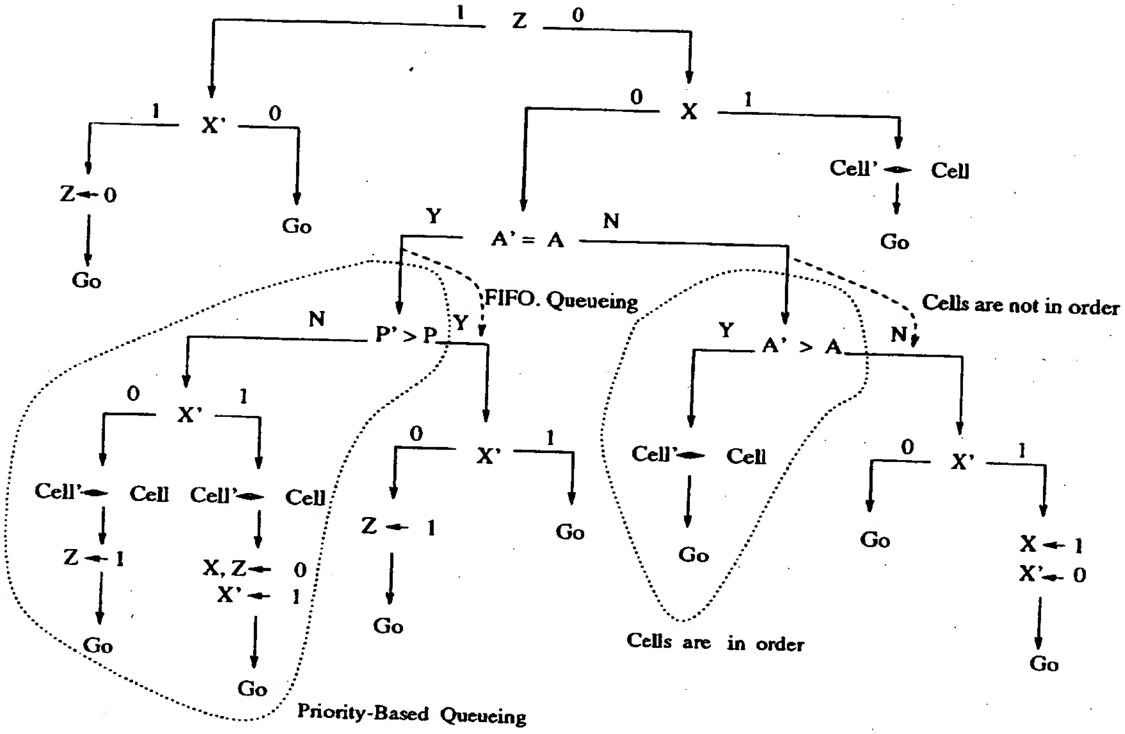


Figure 8: Virtual structure of a logical queue



A: Address, P: Priority, Go: Go to the next unit.
Items with (') refer to the cell in the unit.
Cell ← Cell' : Exchange the cells (including the flag bits)

Figure 9: The complete logical function of the comparison unit

	Conditions	Action
1	Z=1 X'=0	No change
2	Z=1 X'=1	Z=0
3	Z=0 X=0 A' < A X'=1	X=1 X'=0
4	Z=0 X=0 A' < A X'=0	No change
5	Z=0 X=0 A' > A	Cell'↔ Cell
6	Z=0 X=0 A'= A P' > P X'=1	No change
7	Z=0 X=0 A'= A P' > P X'=0	Z=1
8	Z=0 X=0 A'= A P' < P X'=1	Cell'↔ Cell X,Z↔X',Z'
9	Z=0 X=0 A'= A P' < P X'=0	Cell'↔ Cell Z=1
10	Z=0 X=1	Cell'↔ Cell

Items with (') refer to the cell in the unit.
A: Address
P : Priority

Figure 10: The complete logical function of the comparison unit

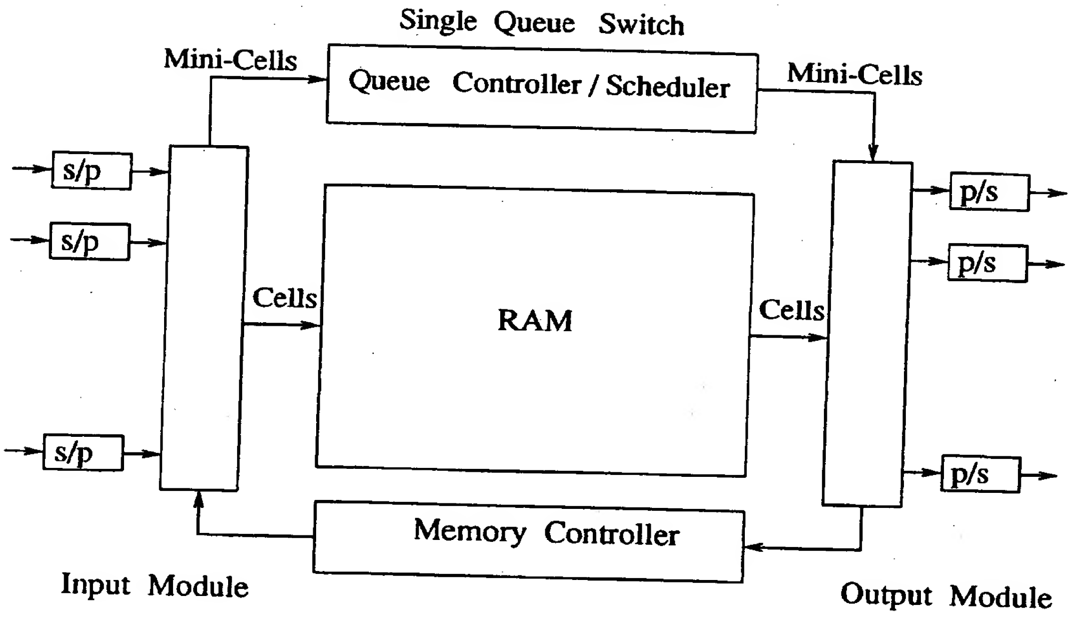


Figure 11: RAM-based ATM switch with the single queue switch as its controller

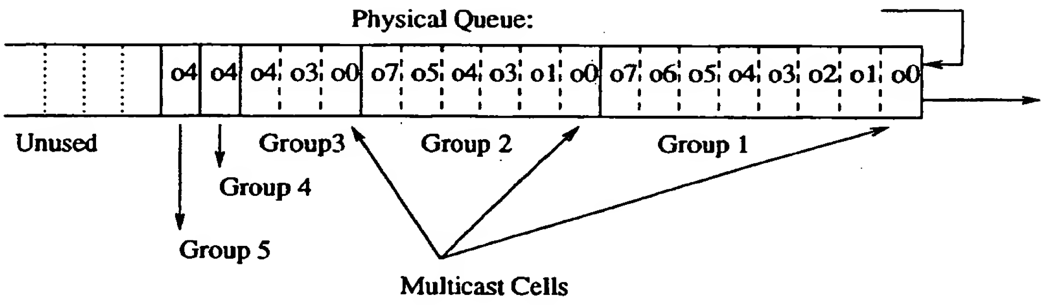


Figure 12: Grouping in multicast single-queue switch

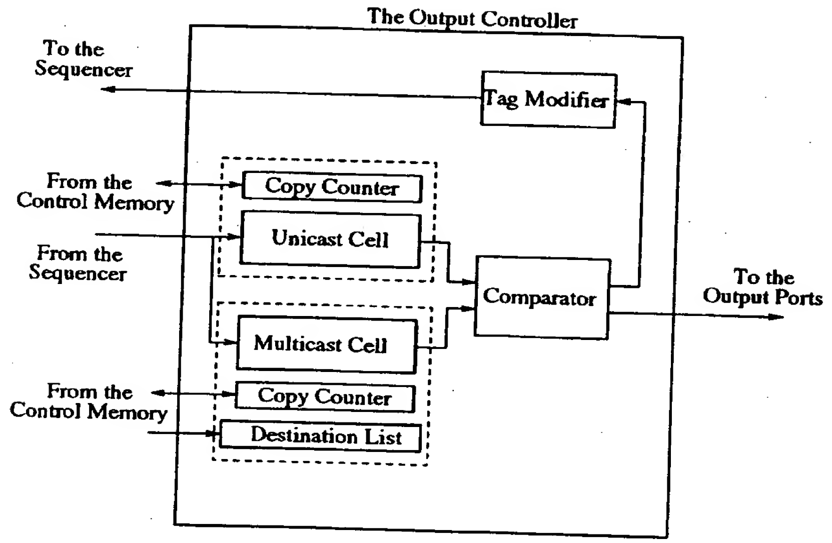


Figure 13: The block diagram of the output controller in the multicast single-queue switch

Minicell

Pointer	R	M	Z	X	Priority	Output Port	V

Flag Bits

Pointer	R	0	Z	X	Priority	1 ~ N	1

A Unicast Minicell

Pointer	R	0	Z	X	Priority	00000	1

A Multicast Minicell

Pointer	R	1	Z	X	Priority	1 ~ N	1

A Copy Cell

V: Validity Flag, X and Z: Grouping Flags, M: Multicast Flag, R: Reserved Flag

Figure 14: The tag format in minicell for unicast, multicast, and copy minicells

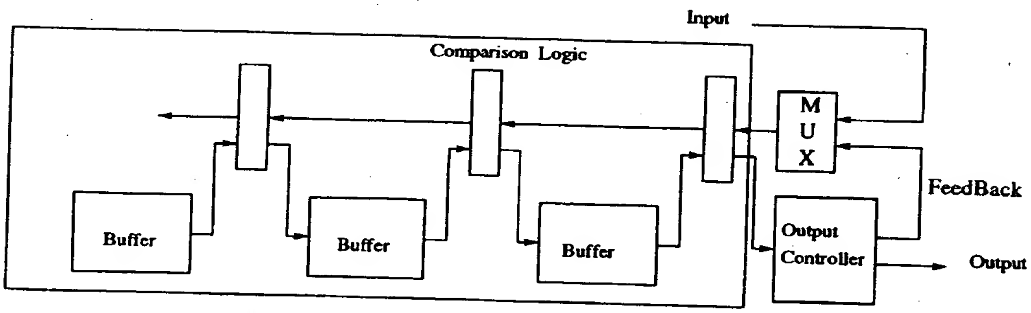


Figure 15: Multicast single-queue switch with a single backward path